Parallel stepwise stochastic simulation: Harnessing GPUs to Explore Possible Futures States of a Chromosome Folding Model Thanks to the Possible Futures Algorithm (PFA)

Jonathan Passerat-Palmbach, Jonathan Caux, Yannick Le Pennec, Romain Reuillon, Ivan Junier, François Kepes, David R.C. Hill

UMR 6158 CNRS

Parallel Stepwise Stochastic Simulation: Harnessing GPUs to Explore Possible Futures States of a Chromosome Folding Model Thanks to the Possible Futures Algorithm (PFA)

Jonathan Passerat-Palmbach [1] [2] * † ‡ Jonathan Caux * † ‡ § Yannick Le Pennec ¶ † Romain Reuillon ‖ Ivan Junier [3] ** François Kepes †† David R.C. Hill * † ‡

Originally published in: SIGSIM-PADS'13, Montréal, Québec, Canada. — May 19–22, 2013 — pp 169–177
978-1-4503-1920-1/13/05

[1] corresponding author

[2] This author's PhD is partially funded by the Auvergne Regional Council and the FEDER

[3] corresponding author

* ISIMA, Institut Supérieur d'Informatique, de Modélisation et de ses Appplications, BP 10125, F-63173 AUBIÈRE
† Clermont Université, Université Blaise Pascal, LIMOS, BP 10448, F-63000 CLERMONT-FERRAND
‡ CNRS, UMR 6158, LIMOS, F-63173 AUBIÈRE
§ Murex S.A.S.
¶ INSA - Rennes
‖ Institute of Complex Systems, Paris Ile de France
** Centre for Genomic Regulation (CRG), Dr. Aiguader 88, 08003 Barcelona, Spain
†† Epigenomics Project and Institute of Systems and Synthetic Biology Genopole®, CNRS, University of Evry

RESEARCH CENTRE
LIMOS - UMR CNRS 6158
Campus des Cézeaux
Bâtiment ISIMA
BP 10125 - 63173 Aubière Cedex
France

Abstract: For the sake of software compatibility, simulations are often parallelized without much code rewriting. Performances can be further improved by optimizing codes so that to use the maximum power offered by parallel architectures. While this approach can provide some speed-up, performance of parallelized codes can be strongly limited *a priori* because traditional algorithms have been designed for sequential technologies. Thus, additional increase of performance should ultimately rely on some redesign of algorithms.
Here, we redesign an algorithm that has traditionally been used to simulate the folding properties of polymers. We address the issue of performance in the context of biological applications, more particularly in the active field of chromosome modelling. Due to the strong confinement of chromosomes in the cells, simulation of their motion is slowed down by the laborious search for the next valid states to progress. Our redesign, that we call the Possible Futures Algorithm (PFA), relies on the parallel computation of possible evolutions of the same state, which effectively increases the probability to obtain a valid state at each step. We apply PFA on a GPU-based architecture, allowing us to optimally reduce the latency induced by the computation overhead of possible futures. We show that compared to the initial sequential model the acceptance rate of new states significantly increases without impacting the execution time. In particular, the stronger the confinement of the chromosome, the more efficient PFA becomes, making our approach appealing for biological applications.
While most of our results were obtained using Fermi architecture GPUs from NVIDIA, we highlight improved performance on the cutting-edge Kepler architecture K20 GPUs.

Keywords: GPU; parallel simulation; chromosome folding; Possible Futures Algorithm; PFA; stochastic simulation; stepwise simulation

1 Introduction

Two major design choices are available for code parallelization. One possibility is to adapt the existing application to fit a parallel template, which mainly consists in breaking loops to spread their workload across several parallel computing elements, be they threads or processors. This approach tends to be limited by the so called Amdahl's law [Amdahl, 1967, Hill and Marty, 2008] that computes the maximum speed-up of an application, taking into account the part of it that can effectively be parallelized. Another possibility consists in redesigning the application from scratch, to use the most possibility of the target parallel platform. In this case, the application is more likely to benefit from performance gain laid down by the Gustafson's law [Gustafson, 1988]. Both situations present advantages: adapting an existing application will make it easier to write; redesigning an application from scratch often opens new parallelization perspectives.

For instance, when dealing with particular hardware accelerators such as GPUs, which are constrained by their underlying architecture, it is usually more efficient to provide a dedicated parallel implementation. Such an approach allows this implementation to better fit the platform's requirements. In the case of GPUs, applications are expected to favour computing intensive algorithms rather than those containing a high number of branches instructions.

Parallelization of an existing code, even from scratch, can be severely limited by the sequential nature of the underlying algorithms, which have been originally thought for Turing-like architectures. In this context, it may be useful to redesign the algorithmic principles themselves, just as new algorithmic schemes have emerged from the possibility of quantum computing. In this article, we present the redesign of a traditional model of polymer folding, which has been extensively used in the field of chromosome modelling [Langowski and Heermann, 2007]. These models are known to suffer from poor efficiency when considered in situations that are similar to *in vivo* because of the strong confinement of chromosomes. So far, "simple" code parallelization has mainly been set up using tools such as OpenMP [Dagum and Menon, 1998] or MPI [Message Passing Interface Forum, 1993]. More recently, parallelization of dynamical models have been realized on GPU architecture, demonstrating the usefulness of such technology for the modeling of long polymers [Reith et al., 2011]. In the same spirit, our algorithm is mainly meant to exploit massively parallel processors, more particularly NVIDIA CUDA-enabled GPUs.

Our algorithm is based on a two-step procedure: i) generation of several potential future states, and ii) selection of the next state as a random choice amongst the set of valid futures. We hence call this approach the "Possible Futures Algorithm" (PFA). PFA can be compared to the branch prediction feature available in modern CPUs, which enables CPUs to load the instructions following the branches of a conditional statement, prior to knowing the result of this statement. This avoids offloading the pipeline mechanism, and consequently waste precious instruction cycles [Lee and Smith, 1984, Smith, 1998]. As a result, we show that PFA highly increases the acceptance rate of the new

model. While the initial algorithm only accepts $\sim 5\%$ of the proposed states in certain configurations, our solution maintains a satisfying acceptance rate over 60% in the same conditions, regardless of the input parameters of the model.

The paper is organized as follows. First, we describe the initial chromosome folding model on which our parallel declination is based. Next, we point out the limitations that led us to design a new model. We then introduce our new parallel chromosome folding model and the PFA approach. Finally, we discuss some implementation choices and study the performances of our algorithm.

2 A classical Monte-Carlo simulation of polymer models

Large-scale properties of polymers have been efficiently captured thanks to simplified models that ignore the details of both the polymer composition and the exact nature of the surrounding solvent [de Gennes, 1988]. In biology, various models have been used to model the behavior of DNA molecules and of chromosomes [Strick et al., 2003]. Chromosomes have been modeled as simple flexible chains that cannot overlap with themselves (self-avoidance effect) – see Figure 1. These so-called worm-like chains provide a coarse-grained description of protein-coated DNA that is simple enough so that, using Monte-Carlo simulations, they can be used to investigate the folding properties of rather long biomolecules [Langowski and Heermann, 2007]. Thus, they have been used to address both the problem of the structuration of chromosomes *in vivo* [Fudenberg and Mirny, 2012] and, more recently, the interplay between structuring and function [Junier et al., 2012].

The worm-like chain is a continuous polymer model that needs to be discretized to be simulated on a computer. To this end, the polymer chain is divided into contiguous cylinders. Two consecutive cylinders are jointed together so that they can rotate freely around their joint (Figure 1), which is used to make the polymer conformations evolve dynamically. To mimic the persistence of the directional order of the polymer chain (see Figure 1), the junctions between each cylinder (hereafter called joints) carry an energy called "bending energy". This energy is exchanged with the solvent using a Monte-Carlo algorithm (see below) and depends on the angle between the two cylinders connecting the joint. We further consider chromosomes that are confined into cells that are more or less narrow, so that our approach can address the computational properties of chromosomes in confined geometries [de Gennes, 1988]. In the following, we call "cell" the embedding volume of the chromosomes during the simulations, *i.e. in silico*. In the most general case, this may not correspond to the nucleus in which chromosomes are embedded *in vivo*.

In summary, a chromosome *in silico* can be viewed as a long string of contiguous "cylinders". The chromosome can be circular (as in bacteria) or linear (as in human). Its statistical properties are the result of its jiggling motion due to the interaction with the surrounding solvent. In these models, the solvent

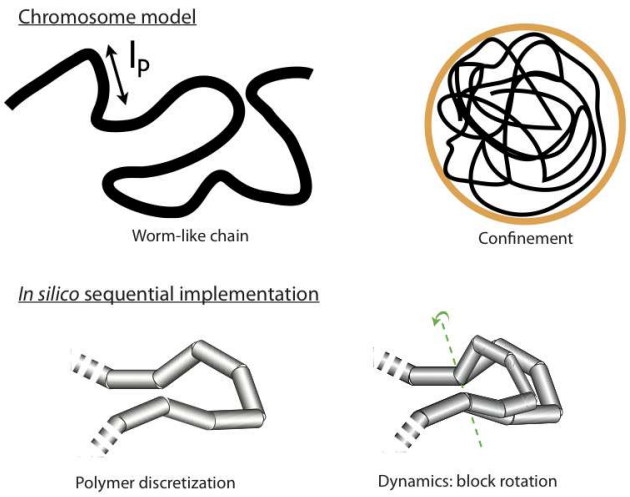

Figure 1: Worm-like chain model of chromosomes. On the top is indicated the original continuous model, which is characterized by the length beyond which the chromosome looses the memory of its directional order (persistence length l_p). The actual situation to be modelled is a situation where the chromosome is confined to a small volume (on the right). To be simulated on a computer, this model is discretized into cylinders (bottom). In this discretized version, two consecutive cylinders can rotate around their common joint but have to pay some energy to do so because as they are semi-flexible (captured by the persistence length). In the simulations, we use five cylinders per persistence length (figures adapted from [Junier et al., 2010]).

is not taken explicitly. Instead, we used a standard stochastic algorithm, also called a Monte-Carlo procedure, to make the chain move step by step. Each of these steps is conceptually simple, and altogether, they form Algorithm 1.

We used the standard Metropolis rate for accepting the rotation depending on its energy, *i.e.* if the new bending energy is lower, the transition is accepted; otherwise, the transition is accepted with probability $e^{-\Delta E/k_B T}$ where ΔE stands for the increase of energy and $k_B T$ for the thermal energy characterizing the solvent (k_B is the Boltzmann constant and T is the temperature). This dynamics is known to satisfy the required properties to simulated the thermodynamic behaviour of polymer models. In practice, it favours movements whose energy variation is dictated by the value of the temperature T. As a result, the chromosome is prompted to a state of equilibrium between the energies of the conformations and the number of equivalent conformations at a given energy, also called entropy.

This stochastic algorithm slowly makes the chromosomes evolve following a combination of random and physical constraints, over hundreds of millions of runs. The limiting factor for this evolution happens when most rotations are

Algorithm 1 Monte Carlo procedure involved in the simulation process

```
Select a section of consecutive cylinders of random size (called "block" in the
following);
Perform a rotation of the block by a random angle around the axis that
intercepts the first and last joint of the block;
if The rotated block does not overlap with the rest of the polymer then
    Compute the new bending energy of the joints at the edges of the block;
    Accept the rotation with some probability depending on the variation of
    the bending energy;
end if
Increase time +1;
```

rejected due to the overlapping conditions, which occurs very often when the embedding cell is small, as in biological situations. In particular, simulations of large chromosomes become extremely time-consuming, which make this type of algorithms poorly efficient to tackle in detail the folding problems of multiple chromosomes *in vivo*.

3 Limitations of the sequential model

3.1 The collisions bottleneck

The original algorithm picks up a "block" at random on the chromosome, and randomly rotated it around an axis. And so on. By definition, it is therefore sequential. As a consequence, for string confinement constraints, *i.e.* for small cell sizes, it is difficult for the algorithm to find valid configurations towards which to evolve. Thus, because valid configurations are hard to find through random attempts, a large number of (wasted) steps is required to provide a new valid configuration. In situation of string confinement, 95% of the generated rotations can produce collisions (see below). It is consequently a major bottleneck that needed to be tackled in order for the model to scale up with chromosomes sizes. A first possibility would be to reduce the amplitude of the rotations. However, in this case, motions of the polymer are very small and both a large number of blocks and a large number of tries are necessary to thermalize the system (*i.e.* to make it visit all most likely conformations). Thus, in any case, the use of parallel motions can facilitate the dynamical evolution of the polymer.

3.2 The Possible Futures Algorithm (PFA)

We propose a parallel simulation model which considers two parallel optimizations. The first one consists in handling several blocks in parallel, instead of only one block as in the sequential algorithm. Processing blocks in parallel is possible in our case because it does not break any physical law at the heart of the model. In particular, it does not break the so-called ergodicity, *i.e.* the

unbiased sampling of the configuration space. This new parallel rotation step is presented in Figure 2.

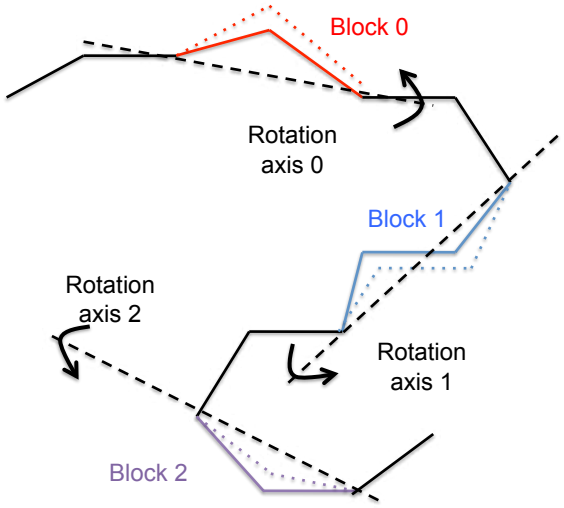

Figure 2: A parallel rotation step

The second optimization consists in considering for each single block a set of several possible rotations to maximize the chances of producing a correct rotation. We call each of these rotations a "possible future" of a block. In Figure 3, we can see a chain of cylinders called "Base". It represents the chromosome as it is at the beginning of an iteration. Yellow cylinders belong to different blocks labelled Block 0 through Block 2. Starting from blocks defined on the base chromosome, the algorithm will generate several possible futures for each block, applying each time a different rotation angle to the block. In Figure 3, three futures are generated for each of the three blocks defined.

Cylinders in grey do not belong to any block, and therefore remain static during this iteration. It is important to note that at each step, the current state of a given block is also viewed as a possible future. Consequently, the fact of not rotating a block is valid, and is always a possible solution for the dynamical evolution of the polymer (which is here a no-motion).

4 Parallel model

The Possible Futures Algorithm induces extra computation time, since futures need to be created before each step of the simulation, and merged as a complete new state for the whole chromosome at the end of each simulation step. However, such an organization reminds of a frequently used pattern in parallel computing: map/reduce [Dean and Ghemawat, 2008]. This pattern is particularly adapted to GPU computing and some frameworks can even simplify its implementation

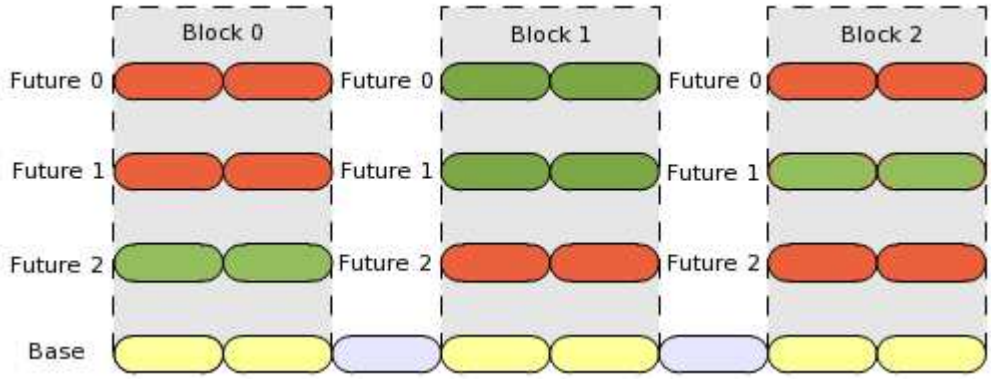

Figure 3: Example of generation of three possible futures for three different blocks

on GPU [He et al., 2008]. In our case, the map stage consists in generating and processing new futures; this stage is processed in parallel, while the reduce stage is performed sequentially to build the resulting state of the chromosome at the end of the current simulation step. Next section will detail the stages of our parallel simulation model.

We describe now the consecutive steps allowing the parallel algorithm to generate and transform possible futures, before combining them as a whole new state for the chromosome. In view of the massive number of repeated parallel tasks that result from the possible future approach, we have chosen to exploit GPU accelerators to run this algorithm.

4.1 Generate possible futures

First, the map stage must be initiated sequentially by generating random blocks in the chromosome. Blocks are equally sized arrays of consecutive joints from the chromosome. Each block owns a different, non-overlapping chain of joints. Blocks also differ from the random angle describing the rotation which will be applied to them. Blocks cannot overlap to prevent the same cylinder from being involved in two different rotations.

To generate futures, a "master" thread runs alone on the GPU to define random blocks and to generate random rotation angles. Then, a burst of threads is launched to clone the original blocks and change their rotation angles, thus making them into possible futures of the original block. Upon completion, possible future blocks are ready to be processed in parallel by thousands of GPU threads.

Calling F the number of possible futures per block and N the number of blocks, one can see that this stage operates in parallel the equivalent of what the sequential algorithm used to do over $F \times N$ time step. The potential performance gain is hence on the order of $F \times N$ since while the sequential simulation

checks after each step that the block can be moved, the parallel implementation fully transforms each possible future without wasting time on intermediate verifications.

4.2 Select valid futures

Once all the futures have been transformed and represent a potential future state of their base block, it is necessary to filter them: *i.e.* dismiss those that are not valid. The algorithm calculates the bending energy carried by the joints at the edges of a future block. To be stated as valid the resulting bending energy of a future must be lower than the one of its original block. Should the new bending energy be greater than the original one, then the considered future has gained energy. In this case, there is a probability that the future may be valid, determined by a random draw according to the physical properties of chromosomes. In any other case, the physical properties of the future make it unsatisfying, and it is consequently withdrawn. The remaining futures must still meet two other requirements, referred to as static and dynamic conditions, to be compatible with their environment and to be potentially included in the global solution.

Static conditions are met when all cylinders of the future are within the virtual cell in which the chromosome must remain confined. Moreover, none of the cylinders of the future can collide with parts of the chromosome that did not evolve during this simulation step. By doing so, we eliminate all futures that are not physically possible.

For dynamic conditions to be valid, a second evaluation determines which futures are suitable with respect to the other futures. A future is suitable as long as it does not collide with any of the other valid future generated at this step of the simulation.

Futures acceptance tables are displayed in green in Figure 3, while dismissed futures appear in red.

To sum up, we have introduced extra operations when delaying the selection of valid futures, but this stage offers many potential solutions to increase the acceptance rate of rotations that make the chromosome evolve. This stage justifies on its own the use of a massively parallel architecture such as GPUs to speed-up the computation of the algorithm. Let's now study the operations which compose the reduction stage of a simulation step.

4.3 Determine compatible futures

We consider a two-round reduction in our algorithm. The first round models the problem as a graph to determine the compatible futures. The vertices of the graph represent possible futures, and the presence of an edge between two vertices represents the compatibility between two possible futures.

4.4 Compose the global result

The second reduction step now aims at composing a global result from the compatible futures identified by the graph. From this graph hundreds of random cliques are built from the whole set of possible futures at hand. A constant number of vertices is simply picked at random in the graph to build equally-sized cliques, with the particular care to always draw a vertex from each group of possible futures.

We now compose a global result by finding one maximum clique from the set of cliques issued previously. A maximum clique owns as many possible block futures as there are blocks in the original chromosome. Thus, any maximum clique forms a solution to our problem and its nodes are the new state of each block defined at the beginning of the simulation step. Please note: should several maximum cliques be found during the first round of the reduction, one would be chosen at random, since they all represent equivalent satisfying solutions to make the chromosome evolve.

The elements from Figure 3 are displayed as a graph in Figure 4. It shows that the possible future 0 from Block 1 is suitable in terms of physics criteria, but is not compatible with any possible future from Block 2 (Although this future is compatible with the current state of the chromosome, for the sake of simplicity, it is not shown in Figure 4).

In Figure 4, we see a clique compounded of 3 vertices:
{*Block0.Future2, Block1.Future1, Block2.Future1*}.
This means these three are inter-compatible futures, and together form a global consistent future for the chromosome. To validate the whole operation, each possible future block is stored in the base chromosome, replacing its original counterpart.

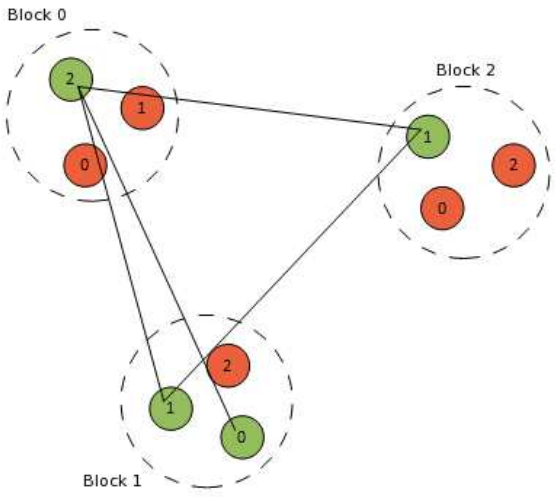

Figure 4: Compatibility graph of the possible futures

5 GPU implementation choices

5.1 GPUContext: avoid memory transfers between host and device

GPU applications often suffer from the excessive latency induced by memory accesses. It is strategic to optimize them as much as possible in order to obtain the best performance (memory transfers from the host to the device, or memory accesses from threads on the device) [Ryoo et al., 2008].

In our case, we have decided to cut any communication between the host and the device, except synchronizations. Indeed, GPUs cannot have all their threads synchronized at one point, at the time of writing, due to hardware constraints. On the other hand, some stages of the simulation need the data to be in a particular state before they can achieve their part of the computation. The only way to synchronize all the threads on a GPU is to wait for the completion of the kernel that has launched them. Each simulation step is consequently divided into several parallel stages, launched consecutively on the GPU. Such a design enables us to finely tune the parallelism grain of the simulation: at each stage the parallelism grain is the most adapted to the kernel that is to be launched.

As a result, communications are shrunk to their utter minimum from the host to the device. Simulations data are transferred back to the host from the GPU only when a checkpoint of the state of the simulation is reached.

5.2 Approximate cylinders to avoid branch divergence

In the original model, the elements that compose the chromosomes are represented by cylinders with rounded edges, as shown in Figure 5. In this diagram, the point *mid* is the middle of the cylinder, and *e1* the direction vector that indicates the orientation of the cylinder. *rin* and *rfin* are two points which denote the edges of the cylinder.

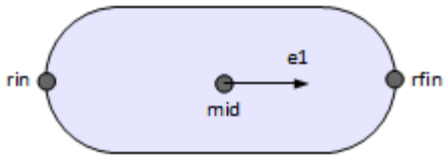

Figure 5: Cylindrical element that is part of the chromosome

This representation has the advantage of being accurate according to the observed physics, but on the other side, it highly complicates the calculation of collisions. Indeed, computing an intersection between two cylinders is rather slow: the corresponding code is about 200 lines and contains many conditions, which makes it less suitable for execution on GPUs.

Instead, cylinders can be approximated by groups of spheres, as illustrated in Figure 6. Points *sphere0*, *sphere1* and, *sphere2* are the centres of the three spheres composing the cylinder. Points *joint0* and *joint1* are the edges of the cylinder.

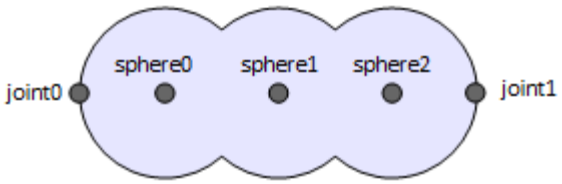

Figure 6: Pseudo-cylindrical approximation of a segment

Thanks to this representation, calculating the presence of a collision between two cylinders consists in calculating the collisions between the three spheres of the first cylinder and the three spheres of the second cylinder. Furthermore calculating collisions between two spheres is a trivial operation, as shown in Listing 1:

Listing 1: Calculation of collisions between spheres

```
bool Sphere::collision (
        const Sphere & otherSphere)
{
  Vector3 vector = this->Position -
             otherSphere.position;
  // All spheres have a radius RADIUS.
  // We compare the squared lengths
  // to avoid an expensive square root.
  return vector.squaredLength() -
       (RADIUS * RADIUS);
}
```

Three spheres (rather than two or four) were chosen because it allows a very good approximation of the original cylinders and collisions observed. Moreover, using more than three spheres does not increase the accuracy of the approximation in a significant way.

6 Results

Thanks to the log files generated by the application, which contain the 3D coordinates of each joint, we can obtain a 3D representation of chromosomes using third-party software such as *gnuplot*. Figure 7 is the output drawing generated by *gnuplot* for a circular chromosome after 100 million iterations.

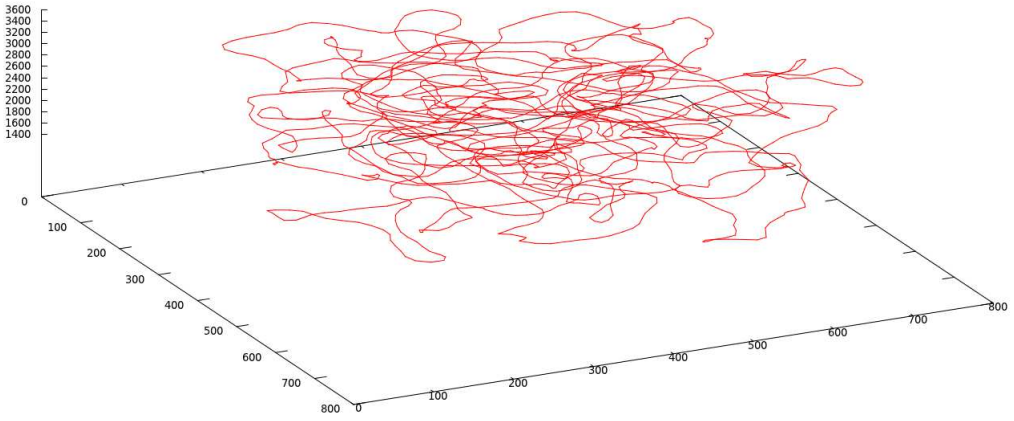

Figure 7: Gnuplot 3D representation of a chromosome

All the results presented in this section have been obtained through simulations run on homogeneous machines equipped with the following configuration: Intel Core i7-2600k 3.40GHz CPUs and NVIDIA GeForce GTX590 GPUs.

6.1 Efficiency of the parallelization approach

The first metric that is interesting to study from the new parallel model is the efficiency of the possible futures approach regarding the acceptance rate. Figure 8, shows the efficiency of the PFA approach in increasing the acceptance rate. As we can see, the acceptance rate increases with the number of possible futures involved in each simulation step.

For clarity's sake, the acceptance rate depicted in Figure 8 is obtained by averaging the acceptance rates of several configurations benefiting successively from 4, 8, 64 and 128 possible futures per block. The configuration test set is as follows:

- number of blocks $\rightarrow \{1; 3; 7; 15\}$
- maximal block size $\rightarrow \{32; 64; 256; 510\}$

Eventually, it appears that the acceptance rate becomes trickier to improve when the number of blocks increases. Indeed, the number of cliques grows exponentially with the number of blocks. Thus, it is more and more difficult for the algorithm to find a maximal clique amongst the set of randomly generated cliques.

6.2 Performance on the cutting-edge Kepler architecture K20 GPU

Parallel architectures and especially GPUs evolve very quickly. In the particular case of NVIDIA GPUs, a new generation is shipped every two years. Thus,

an application running on this architecture must take advantage of these evolutions and display improved performance when run on the forefront GPUs. Having run our benchmark on Fermi architecture GTX 590 GPUs, we have tested some identical configurations on a recent Kepler architecture K20 GPU [NVIDIA, 2012]. Figure 9 depicts the speed-up obtained at no cost, only by switching from the GTX to the K20 GPU.

7 Conclusion

In this paper, we have described a parallel model of chromosome folding and its implementation on GPU, using the NVIDIA CUDA technology.

The purpose of this model is to make chromosomes evolve in the confined space of a cell, by applying successive rotations on random chunks of the chromosome. This new parallel model is based on an initial sequential one, which encounters difficulties when the cell space becomes more confined. The acceptance rate of the simulated rotations then quickly falls, making the sequential model evolve slower.

In order to increase the acceptance rate of simulated rotations, especially when the space becomes more and more confined, we designed a Possible Futures Algorithm (PFA). This approach proposes different new configurations for each block at each step, which have to be checked for inter-compatibility. All the futures can be generated and processed in parallel, thus inducing a massively parallel workload of repetitive tasks.

This approach has shown satisfying results in terms of acceptance rate. While the original model was likely to dismiss about 95% of the rotations, the possible future approach displays a satisfying acceptance rate of more than 60%. Such an acceptance rate enables the chromosome to evolve to a new configuration after almost each simulation step.

We now plan to challenge this parallel model with an implementation of the sequential one. Such a benchmark will have to be conceived very carefully, as long as the two models operate in two different ways and the same metrics cannot be easily compared between the two models. For instance, a simulation step does not perform the same actions in the two models. That is why we focused on the acceptance rate in this study. Still, the parallel implementation should display far better performance than the sequential one thanks to its possible futures approach. The more the chromosome becomes confined within the cell space, the more the parallel algorithm will be efficient. In fact, the efficiency of the parallel implementation is due to its ability to maintain a high acceptance rate throughout the execution, whereas the sequential model will see its acceptance rate decrease with the various stages and the confinement.

The interested reader will find a free version of both sequential and parallel implementations in the git repository of the French Institute of Complex Systems[1].

[1] `https://forge.iscpif.fr/projects/dna/repository`

8 Acknowledgments

Ivan Junier is supported by a Novartis grant (CRG). This work is also partly funded by the Auvergne Regional Council and the FEDER. The authors would like to thank Alice Chalet for her careful reading and useful suggestions on the draft.

References

[Amdahl, 1967] Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM.

[Dagum and Menon, 1998] Dagum, L. and Menon, R. (1998). Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55.

[de Gennes, 1988] de Gennes, P.-G. (1988). *Scaling concept in polymer physics*. IthacaNY: Cornell University Press, third edition.

[Dean and Ghemawat, 2008] Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

[Fudenberg and Mirny, 2012] Fudenberg, G. and Mirny, L. A. (2012). Higher-order chromatin structure: bridging physics and biology. *Current Opinion in Genetics & Development*, 22(2):115–124.

[Gustafson, 1988] Gustafson, J. L. (1988). Reevaluating amdahl's law. *Communications of the ACM*, 31(5):532–533.

[He et al., 2008] He, B., Fang, W., Luo, Q., Govindaraju, N. K., and Wang, T. (2008). Mars: a mapreduce framework on graphics processors. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 260–269. ACM.

[Hill and Marty, 2008] Hill, M. D. and Marty, M. R. (2008). Amdahl's law in the multicore era. *Computer*, 41(7):33–38.

[Junier et al., 2012] Junier, I., Dale, R. K., Hou, C., Képès, F., and Dean, A. (2012). CTCF-mediated transcriptional regulation through cell type-specific chromosome organization in the -globin locus. *Nucleic Acids Research*.

[Junier et al., 2010] Junier, I., Martin, O., and Képès, F. (2010). Spatial and topological organization of dna chains induced by gene co-localization. *PLoS computational biology*, 6(2):e1000678.

[Langowski and Heermann, 2007] Langowski, J. and Heermann, D. (2007). Computational modeling of the chromatin fiber. *Seminars in Cell & Developmental Biology*, 18(5):659–667.

[Lee and Smith, 1984] Lee, J. K. and Smith, A. J. (1984). Branch prediction strategies and branch target buffer design. *Computer*, 17(1):6–22.

[Message Passing Interface Forum, 1993] Message Passing Interface Forum (1993). Document for a standard message-passing interface. Technical report, University of Tennessee.

[NVIDIA, 2012] NVIDIA (2012). Nvidia's next generation cuda compute architecture: Kepler gk110. Technical report.

[Reith et al., 2011] Reith, D., Mirny, L., and Virnau, P. (2011). Gpu based molecular dynamics simulations of polymer rings in concentrated solution: Structure and scaling. *Progress of Theoretical Physics Supplement*, 191:135–145.

[Ryoo et al., 2008] Ryoo, S., Rodrigues, C. I., Baghsorkhi, S. S., Stone, S. S., Kirk, D. B., and Hwu, W.-m. W. (2008). Optimization principles and application performance evaluation of a multithreaded gpu using cuda. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 73–82. ACM.

[Smith, 1998] Smith, J. E. (1998). A study of branch prediction strategies. In *25 years of the international symposia on Computer architecture (selected papers)*, pages 202–215. ACM.

[Strick et al., 2003] Strick, T. R., Dessinges, M.-N., Charvin, G., Dekker, N. H., Allemand, J.-F., Bensimon, D., and Croquette, V. (2003). Stretching of macromolecules and proteins. *Reports on Progress in Physics*, 66:1.

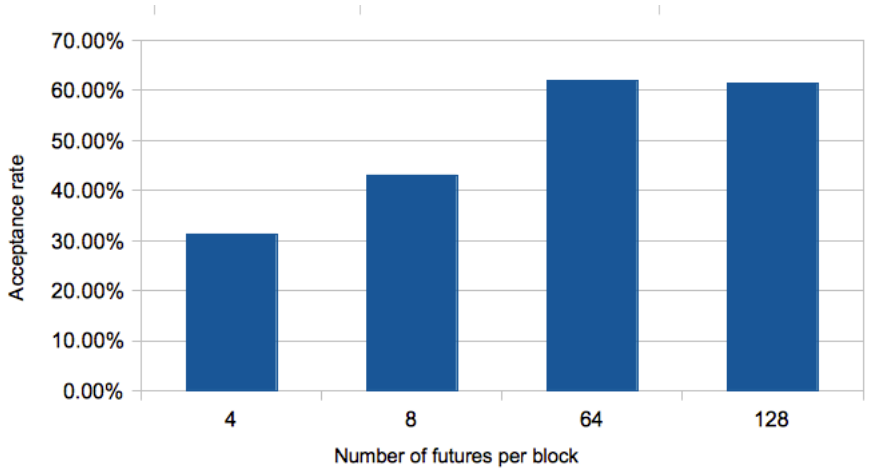

Figure 8: Increase in the acceptance rate with the number of possible futures

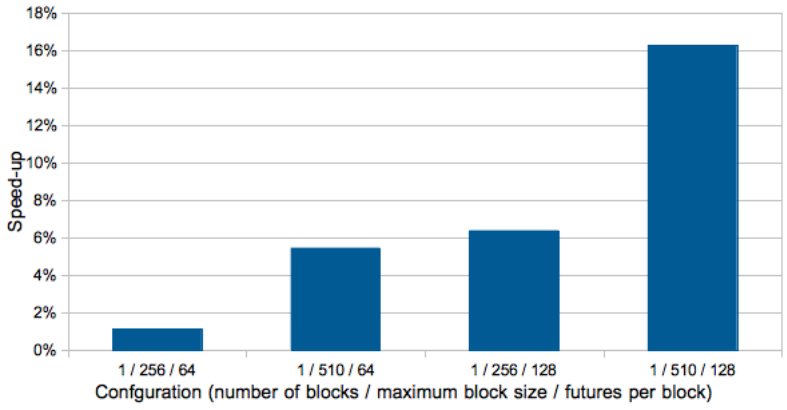

Figure 9: Model improved performance when run on a Kepler architecture K20 GPU

UMR 6158 CNRS

RESEARCH CENTRE
LIMOS - UMR CNRS 6158
Campus des Cézeaux
Bâtiment ISIMA
BP 10125 - 63173 Aubière Cedex
France

Publisher
LIMOS - UMR CNRS 6158
Campus des Cézeaux
Bâtiment ISIMA
BP 10125 - 63173 Aubière Cedex
France
http://limos.isima.fr/